

IMPROVEMENT OF SOFTWARE QUALITY: INTRODUCING EXTREME PROGRAMMING INTO A PROJECT-BASED COURSE

Yael Dubinsky
Department of Computer Science
Adjunct Lecturer
yael@cs.technion.ac.il

Orit Hazzan
Department of Education in Technology and Science
Senior Lecturer
oritha@tx.technion.ac.il

Technion – Israel Institute of Technology.
Technion City, Haifa 32000, Israel
Phone: 972-4-8294323 Fax: 972-4-8293943

Abstract

Different software development methodologies and quality assurance methods are used in order to attain high quality, reliable, and bug free software. eXtreme Programming (XP) is a software development methodology that integrates many of the known ideas (that we all were familiar with) in order to achieve such software systems. Specifically, XP emphasizes code-unit testing (preferably before its writing), and thorough testing of software functionality. This paper presents our experience of teaching XP to Computer Science students. Students' conceptions of quality issues are highlighted.

1. Introduction

Software development is a mentally complicated task. The maintenance of existing and running software is even more complicated and should be considered during the development process. In order to ease software maintenance, software should be developed in a way that enables scalability and the ability to make future changes. This implies that software development process should be an organized, but flexible, process that leads to functional versions and simple software code. If this was easy to achieve, quality assurance would not have been a problem. However, as it is well known, this is not the current situation.

At the beginning of this year (April 2002), an *InformationWeek* research survey was conducted about applications quality and its impact on corporate operations¹. 800 business-technology professionals have responded, who head up application development or serve as managers in IT organizations. The results indicated that 97% of the participants reported problems with software bugs and flaws in the past year, and 9 out of 10 participants reported about higher costs, lost of revenue, or both, as a result. Managers who were interviewed said that software failures harm their business, and that poor software quality is no longer tolerated.

Our paper has two main goals, both with emphasis on quality assurance: first, in section 2 we introduce eXtreme Programming (XP), which is one of the agile methodologies for software development; second, in sections 3 and 4 we illustrate how XP has been implemented in a project-based course at the Department of Computer Science at the Technion and present preliminary results of our study. We emphasize students' attitude towards software quality issues.

2. eXtreme Programming

The contribution of XP to software development is expressed, among other ways, in the quality improvement of both the entire process of software development and of the software quality itself. Currently, XP is used mainly in small-medium size software projects and is enhanced by companies like Rational Software, RADSoft, Oak Grove Systems and others (for more companies look at the 2002 XP Agile Universe Conference at <http://www.xpuniverse.com/sponsors>). According to Breznitz (2002), one of the "distinctive features of the Israeli software industry [that led to its success] was the prominent role played by small-to-medium size firms marketing their own innovative products." (p. 10). Thus, we believe, our study may contribute significantly especially to the Israeli software industry.

XP is a software development discipline in the family of agile methodologies. It guides software development in an incremental process. XP is based on dozen practices, each of them aims to improve software quality. The next two paragraphs outline briefly these 12 practices (presented in *Italics*). Further details can be found in Beck (2000) and the reference list at the end of his book.

¹ The survey results were published in <http://www.informationweek.com/story/IWK20020517S0010> .

XP programmers work *40-hour week*, on frequent software iterations (of about 3-4 weeks) that lead to *small releases* (4-6 months each). At each stage, the design process is based on the general design of the system, but focuses only on those items needed for the current iteration. This leads, eventually, to *simple design*. The XP team members use a *metaphor*, a common collection of names and descriptions that guides development and communication. *Continuous integration* – that is, the integration of each small part of the code by adding it to the system and running the full suit of automated tests – provides rapid feedback to the programmer and, if needed, small changes are performed in the code. The team members perform *testing* operations, and periodically code *refactoring*, that its goal is to improve and simplify the code and its design. The *pair programming* practice (Williams and Upchurch, 2001; Sanders, 2001) – that is, code production is conducted by two programmers working together – ensures that each line of code is inspected by at least two persons. There is a turnover of pairs during development according to developers’ knowledge and requirements. The code complies *coding standards* used by the team, and is *collectively owned* by the team and not by the individuals. This means that more programmers view, simplify, and inspect the code, besides the original pair who wrote it. This increases the teammates’ commitment to the process. The ideal situation of XP is that the *customer* is always *on site*. The customer has an important role in the *planning game* where the decision is made with respect to the scope of each release and iteration. According to this determined scope, the development work is estimated and divided with load balance among the team members.

High software quality is one of the main principles that guide any software development methodology. This is also correct with respect to XP. For this aim, XP defines two levels of automatic testing (Crispin 2001). The first is *unit testing*, which must be coded by the programmers, preferably right **before** coding a given feature. In contrast to the traditional software development methods, where the quality control entity checks the software quality and provides feedback to the programmers, in the XP way, the quality control is part of the development process and the feedback is given within the team itself. The second level of testing is *functional testing*. Each *customer story* is tested by at least one functional test. The customer understands each test and provides the values to be tested. The tests are updated when incremental changes are performed. The automated unit testing and the functional testing increase software quality assurance and create a rapid and confident developing process (Jeffries 1999). Furthermore, as it turns out, this testing approach has direct implications on software cost.

Table 1 maps the XP practices, distinguishing the XP practices which address the software quality and those which address the development process quality. This mapping highlights the different aspects concerning quality with respect to XP practices. Furthermore, Table 1 shows that the XP practices, which address the software quality, emphasize technical and code-oriented aspects, where the XP practices, which address the development process, emphasize human and social aspects. Needless to say that both aspects are important and there is a synergetic relationship among them.

Quality aspect Influence level	XP practices address the software quality	XP practices address the development process quality
High	Simple design Testing Refactoring Continuous integration	Planning game Customer on-site Pair programming Collective ownership
Normal	Small releases Coding standards	Metaphor 40-hour week (adopted to academic schedule)

Table 1: XP practices mapping with respect to quality subjects

3. ‘Operating Systems Projects’ Course

The Department of Computer Science at the Technion is considered to be as one of the top ten Computer Science departments in the US. One of the advanced courses that the department offers is the ‘Operating Systems Projects’ course, which the first author teaches for the last six years. This is a project-based course which is learned after two theoretical courses – Introduction to Operating Systems and Operating Systems Structure. Our study examines the integration of XP into the course with some studio elements. The studio is the basic learning method used in architecture schools and it has been adopted by some other design disciplines as a training environment. In such studios, students develop projects with close guidance of a tutor and with on-going reflection, both on what is created and on the creation process.

Studio descriptions in Software Engineering education can be found in Tomayko (1996) and in Kuhn (1998). Analysis of how the studio can be implemented into software development environments is presented by Hazzan (in press, 2002).

Prior to the course, a team of four supervisors was introduced to XP and trained to teach it. The first author, who was the instructor in charge of the course, led the supervisor team sessions. XP was introduced and the supervisors experienced some of the practices. Main issues of the course were discussed, including the structure of the studios, the roles of the supervisors in the studio, the special roles of the students, and the evaluation model. All decisions were made jointly by the supervision team. In the 2002 summer semester, XP was implemented in the form of four studios. Each studio worked on a different operating system project, chosen by its supervisor, in teams of 10-12 students. Throughout the semester XP practices were taught and employed. Attendance to all sessions, 4-hours meeting each week, was compulsory.

4. Study Results

Throughout the semester data was collected by the following research tools: questionnaires, video tape of studio sessions, frequent conversations with students and supervisors, students' weekly reflections (to be submitted at the end of each session), and team diaries. Naturally, the different research tools provided us with observations of different kinds. This section brings data obtained mainly from students' weekly reflections. For the purpose of the quality conference, we focus on students' conception of software quality.

In addition to the project development in the course, the students had additional assignments. One of this assignments, given to the students towards the end of the semester as a weekly reflection, after the students had learned XP in general and testing in particular, asked students (a) to define software quality and (b) to describe how XP influences the software quality in their team. It is important to note that though the students knew that these weekly reflections are used only for checking their attendance in the meeting and would not affect their final grade, they addressed them very seriously.

In what follows, representative student definitions for software quality are presented. These definitions are quoted without editing students' formulation. Each of the following definitions has its own uniqueness (underlined). The rest of the students' answers address similar issues.

- Software quality is the ability of software to be: readable as much as possible, short and elegant as much as possible, with no bugs, robust – check data received from other modules and return values from functions of other modules.
- It should be correct and then performance should be boosted.
- Software quality is in my opinion, simplicity, reliability, user-friendliness and correctness.
- Software quality from my point of view is a user-friendly software, with no bugs, which we can add features, well documented, and which is written simple and clear.
- Software quality from my point of view is a software that its development time is reasonable, structured in a modular way, and readable enough so another developer can use or change it fast and easy.
- Software quality from my point of view is writing readable, simple, arranged and clear code, well documented, with no bugs, future expendabilities are possible in a relative simplicity, and of course should answer the functionality of the demands.

With respect to the question of how XP influences the software quality in their team, all students wrote that XP has positive influence on the software quality of their team. All together, the students mentioned all the XP practices which were emphasized during the semester as practices that contribute to higher software quality. As they were encouraged to reflect on their negative feelings as well as on the positive ones, they mentioned constraints as well as difficulties with using XP. Following are sections taken from some representative answers.

- ... XP method increases the software quality. The planning game we made is a very useful method, since every time we plan the next work for some short iteration, and ... The XP method makes it possible for us to present something work only two weeks after working.

- When working in pair, one is watching and this way we can provide a more effective program. When working in the team, everyone gives suggestions and this improves the software quality. This way we can reach problems in an earlier stage, than in the case of someone who works alone. The test-first method avoided future problems, since we first think about the testing program, make sure it is right, and only then build the program itself.
- The project wasn't managed according to all XP characteristics. According to XP, the team members should be involved daily in the project. As a large team, whose members are not working daily together, there was a great difficulty to coordinate between the different couples.
- ... another thing is the adhering to customer stories that ensures that the customer will indeed receive what he asked for. The test-first method causes considering things in the code before writing it (reveal problems before the code is written). Also, ensures that for each written code there is a test code (usually we neglect testing due to lack of time in projects) which increases the quality of the software we produce.
- There is a conceptual problem to perform test-first due to old habits. There is no doubt that multiplicity of tests increases and improves the software quality and avoids bugs.
- It [XP] gives us more confidence that the software that we are building is working properly.
- ... the code refactoring adds effectiveness.
- XP contributes to maximum performance, since it is easier to identify bottle-necks, than in the traditional programming method.
- ... and by it's [XP] concept of gradual integration and gradual adding of new features. Also all those qualities are improved by the team work which allows strength-sharing.

As the above quotes indicate, the students know what software quality means. Most of them conceive of XP as a methodology that can improve software quality by its small releases and periodical integration, its planning game which organizes and divides the work among team members, by working both in pairs and as part of a team. All students appreciate the importance of testing (and especially the XP practice of test-first) and of code refactoring. Due to the academic constraints, students complain that XP does not fit completely to academic frameworks since they cannot meet every day and have to coordinate meetings and other issues by electronic tools. Still, some say that this way of course schedule, 4-hours meeting each week with tight scheduled tasks, forced them to work on the project for the entire period of the semester.

5. Conclusion

New learning theories emphasize the importance of students' active involvement when learning new topics (this idea is expressed colloquially by "what students hear they forget, what they see they remember, and what they perform they understand"). Accordingly, we may conclude that our students better understand today the XP methodology of software development. They have experienced working the XP way in an environment which simulates as much as possible the real world conditions and that emphasizes quality assurance of both the development process and of the product itself. Based on previous supervision experience of students' projects, it seems that XP improved significantly the development process in our academic environment. We hope to present further results from the analysis of the data gathered by the other research tools.

References

- Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.
- Breznitz, D. (2002). The Military as a Public Space – The Role of the IDF in the Israeli Software Innovation System, *Report STE-WP-13, Samuel Neaman Institute, Technion – Israel Institute of Technology*.
- Crispin, L. (2001). Is Quality Negotiable?, *XP Universe Conference*.
- Hazzan, O. (In press, 2002). The reflective practitioner perspective in software engineering education, *The Journal of Systems and Software*.
- Jeffries, R.E. (1999). Extreme Testing – Why Aggressive Software Development Calls for Radical Testing Efforts, *Software Testing & Quality Engineering*, pp. 23-26.

- Kuhn, S. (1998). The software design studio: An exploration, *IEEE software* **15**(2), pp.65-71.
<http://www.computer.org/software/so1998/s2065abs.htm>.
- Sanders, D. (2001). Student Perceptions of the Suitability of Extreme and Pair Programming, *XP Universe Conference*.
- Tomayko, J. E. (1996). Carnegie-Mellon's Software development Studio: A five-year retrospective, *SEI Conference on Software Engineering Education*.
<http://www.contrib.andrew.cmu.edu/usr/emile/studio/coach.htm>.
- Williams, L. and Upchurch, R.L. (2001). In Support of Student Pair-Programming, *Proceedings of SIGCSE Technical Symposium on Computer Science Education*, pp. 327-331.